JANA2: Multithreaded Event Reconstruction

D Lawrence[1], A Boehnlein, N Brei and D Romanov

Thomas Jefferson National Accelerator Facility. Newport News, Virginia, USA

E-mail: [1]davidl@jlab.org

Abstract. JANA2 is a multi-threaded event reconstruction framework being developed for Experimental Nuclear Physics. It is an LDRD[1] funded project that will be the successor of the original JANA framework. JANA2 is a near complete rewrite emphasizing C++ language features that have only become available since the introduction of the C++11 standard. Features such as shared pointers, language native threading, and atomics are employed. This paper outlines the status of the project.

1. Introduction

The large volumes of data produced in modern Nuclear and High Energy Physics experiments require large compute resources to analyze. Many-core CPUs are now the norm at the large computing facilities and even university based sites that are used for this analysis. Both logistical concerns and memory limitations motivate a multi-threaded design of the analysis/reconstruction software. JANA (**J**Lab **ANA**lysis framework) was one of the earliest implementations of a multi-threaded, event processing framework designed to run on commercial CPU hardware[1]. The framework has been used for many aspects of the GlueX experiment[2] at Jefferson Lab which was commissioned in 2016. In addition to the reconstruction of both acquired and simulated data, JANA is used for the online monitoring system and was also used to develop a L3 trigger (also known as a high level software trigger). With over a decade of experience in developing and using JANA, an effort has begun to develop a next-generation framework, JANA2. JANA2 leverages language features introduced in the C++11 standard[3] to produce a framework that can help carry the next generation of Nuclear Physics experiments in the coming decade.

2. The JANA Factory Model

From the user's point of view JANA implements a type of factory model where data objects are the product and the algorithms that produce them are the factories. Figure 2 illustrates the analogy to industry. An end user will request a type of data object for the current event from the JANA framework. This request will be matched to an algorithm (*factory*) that can produce it. It will then check to see if the factory has produced the objects of that type for the current event already (i.e. are they in stock). If so, pointers to the const objects are returned to user. If not, then the objects are created first and then the pointers to the const objects are returned. The pointers are to *const* objects to prevent the user from changing the contents of the objects themselves. This is necessary to guarantee that the order in which different user code requests

[1] Lab Directed R&D. See https://www.jlab.org/research/ldrd

the objects cannot affect the object content. For example, two plugins are attached to a process that produce two sets of histograms. If both plugins were to request objects of type A and the first plugin were to modify the values in those objects before the second plugin saw them, then the resulting histograms would differ depending on the presence of the first plugin and even the ordering of plugins.

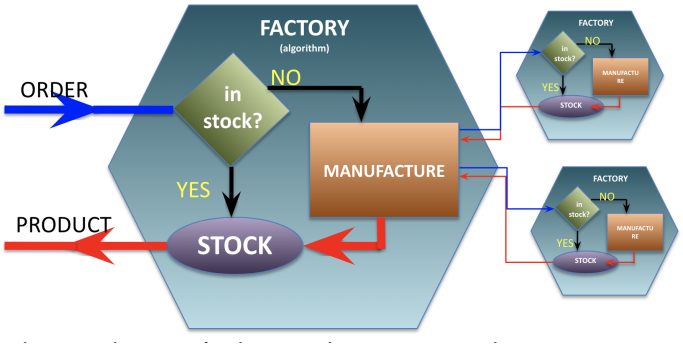

Figure 1. Illustration of JANA's factory model. In JANA, a factory represents an algorithm that produces objects of a single type. The model is designed such that the factory will produce its objects only once for a given event making it very efficient even when mixing plugins from multiple users.

A second feature of the framework is that factories are activated for an event only on demand. This eliminates unnecessary computations for jobs that do not require certain factories to be activated. This model also frees the end user from having to specify which factories should be activated before starting the job. A user may write an event processor that requests only a few mid-level objects. The factories that produce those objects will request the lower-level objects they need and so on and so on. This also allows for very efficient implementation of L3 (i.e. high level) triggers. A decision making algorithm could be designed that asks for low level objects first in order to make a trigger acceptance or rejection decision quickly. If a decision can't be made, then higher level objects would be requested for that event so they could be used to make the decision. This ability to determine which factories should be activate on an event-by-event basis will increase the average throughput of the L3 system requiring fewer computers to implement.

To increase overall performance of the system, a complete set of factories is created for each processing thread JANA creates. This means that each thread has the ability to fully reconstruct an event without requiring interaction with other threads. This minimizes the use of locks and mutexes which limit multi-threading efficiency. Figure 2 provides an illustration. Note that JANA processing threads are created once at the beginning of the job and live throughout the life of the process. A new feature of JANA2 is the ability to easily create or suspend processing threads in the middle of a job. This feature could be done by a user manually or by an automated system as a variable by which it can dynamically optimize resource usage on the compute node.

3. Event Disentangling and Event Level Parallelism

Experimental Nuclear Physics (NP) and High Energy Physics (HEP) share many of the same challenges. One distinction is that the size of the events produced in a typical NP experiment are much smaller than those in an HEP experiment. For example, the GlueX experiment produces events that are about 12kB in size while CMS records events roughly 100 times larger[4]. The data rates are comparable though with GlueX recording events at 90kHz compared to CMS' $\sim$1kHz[5]. The consequence of this is that HEP experiments are motivated to incorporate sub-event level parallelism due to the large amount of memory that is required to process a single event. GlueX requires less than 100MB of memory for each thread[2] (where 1 thread = 1 event). Assuming the memory scales roughly with event size, a $\times$100 size event would require on the order of 10GB per thread. This does not match well with the existing HPC facilities which

[2] Typical GlueX reconstruction jobs have a base memory requirement of 5.75GB with 66MB per thread.

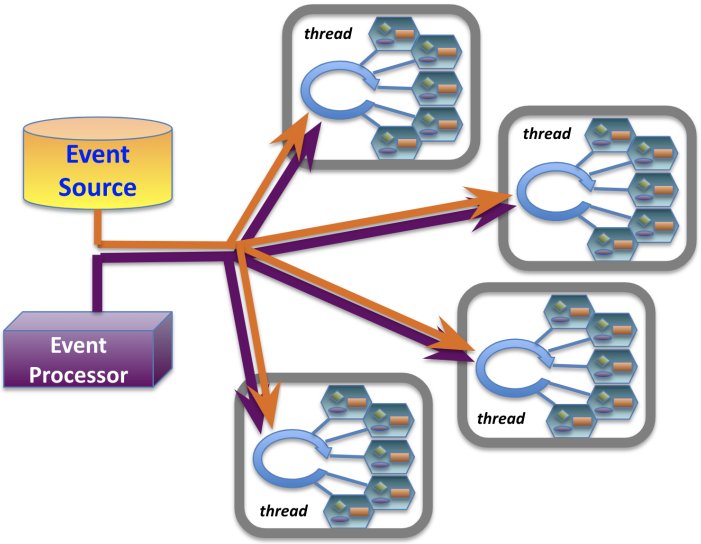

Figure 2. In JANA2 (as in JANA), each thread has its own complete and independent set of factories capable of fully reconstructing an event within that thread. This minimizes the use of locks which would be required to coordinate between threads and subsequently degrade performance. A new feature of JANA2 is that factory sets are maintained in a pool and are (optionally) assigned affinity to a specific NUMA group. As shown in the illustration, events must be read in from the source sequentially(orange) and similarly written sequentially to the output(violet). JANA2 allows reading simultaneously from multiple sources.

tend to equip 1GB-2GB per core. The downside for NP is that the small event size and high trigger rate requires bundling of event fragments at the front end. This leads to the data being recorded in blocks of entangled events where the bundled fragments of different detectors are interleaved. To process the data, a single block of events (typically 40) must be read from the file and disentangled into individual physics events prior to assigning a thread for processing. This is a one-to-many problem where the task of disentangling one event bundle leads to 40 subsequent tasks of processing the 40 individual physics events. This process is illustrated in Figure 3. For GlueX using the original JANA, the parsing tasks were handled by threads not managed by JANA while event reconstruction tasks were managed by JANA. JANA2 improves upon this by generalizing the thread pool to work with tasks assigned to various queues. The one to many problem is handled naturally by tasks being popped from one queue and any number of resulting tasks pushed onto the next queue.

4. Scaling performance

Like the original JANA, JANA2 is being developed to efficiently multi-thread on many core CPUs. Figure 4 shows the results of a test where just the event parsing code for raw data from the GlueX experiment was done. The computer used for the test consisted of 36 physical cores with 36 additional hardware threads (*hyperthreads*) (dual socket Intel Xeon E5-2697 v4). The plot indicates very good scaling for the first 36 threads as well as scaling at a different rate for the 36 additional hardware threads as expected. The red line represents the result of fitting the model defined in the top left of the plot to the data. The serial part of the job is typically dominated by the I/O while the parallel is due to work done in the factories. The plot shown was made for just event parsing so T_{para} is much smaller than one would see for a typical reconstruction job. For full reconstruction, the overall rate would therefore be lower, but the scaling would actually only improve. The lower right corner of the plot gives the Amdahl p value as well as speedup(s) and theoretical upper limit on the rate given the T_{serial} and T_{para} values if one had a similar processor with an infinite number of cores.

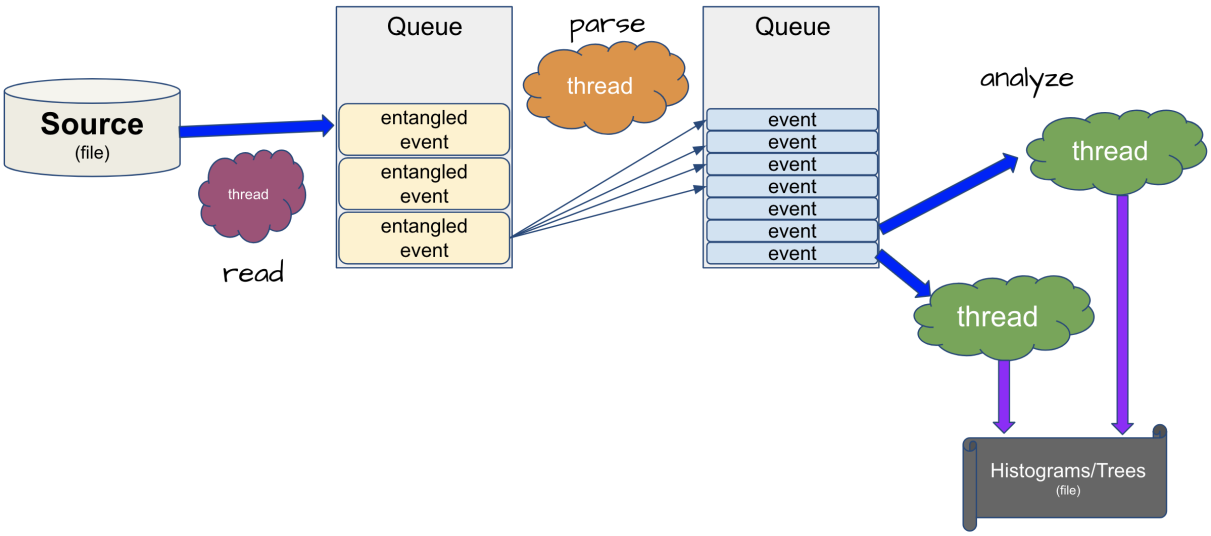

Figure 3. Illustration of how data is moved through queues in modern Nuclear Physics Experimental Data processing. JANA2 improves on JANA by using a single generic pool of threads to perform all tasks removing one level of optimization required by the user.

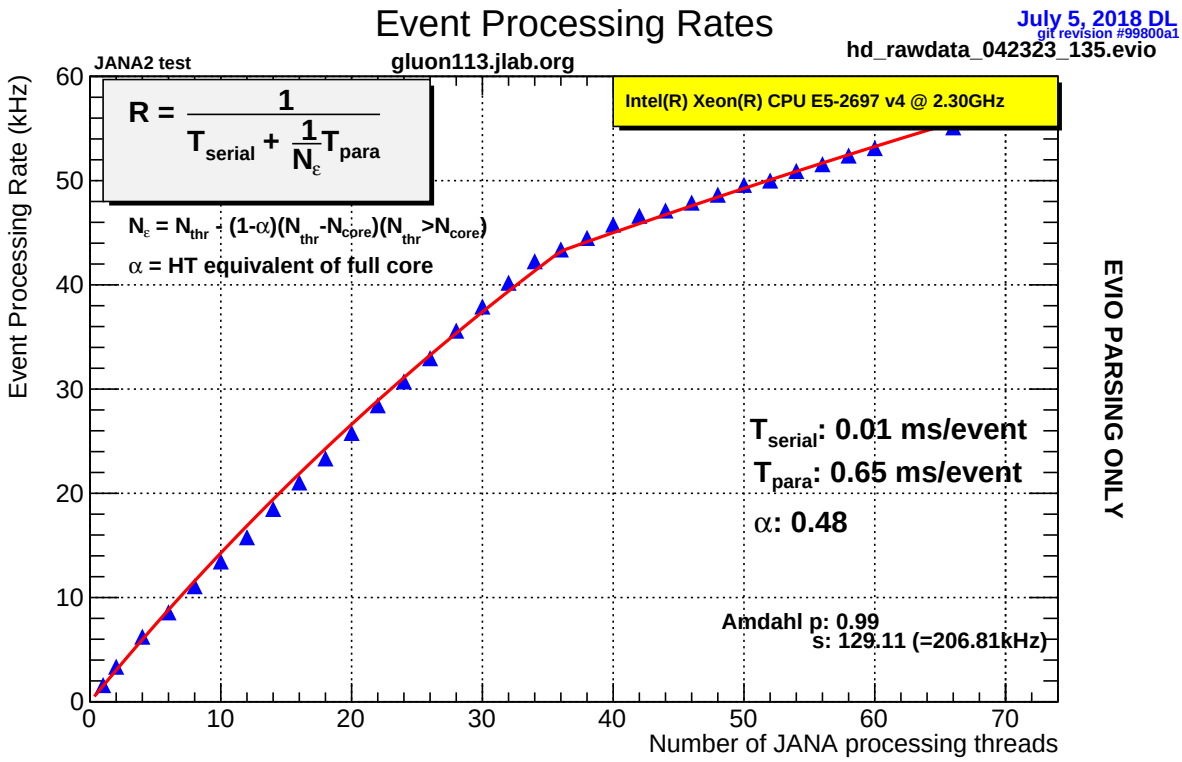

Figure 4. Scaling performance of early version of JANA2. CPU task is parsing of a raw data file from the GlueX Spring 2018 data set. The parsing rate vs. number of threads is shown as the blue triangles. The red curve is a fit to the model shown in the upper left hand corner. See text for more details.

5. Summary

JANA2 is a second generation multi-threaded event processing framework currently being developed at Jefferson Lab. New features are being added that take advantage of recent additions to the C++ language standard as well leveraging experience gained from over a decade of using JANA in the GlueX experiment. The first production release is expected to be available by the end of 2019.

Acknowledgments

This material is based upon work supported by the U.S. Department of Energy, Office of Science, Office of Nuclear Physics under contract DE-AC05-06OR23177.

This work is being done under JLab LDRD project *LDRD*1908

References

[1] D. Lawrence 2008 *Multi-threaded event processing with JANA* (*PoS* no 062) (http://pos.sissa.it/archive/conferences/070/062/ACAT08_062.pdf: SISSA)

[2] Dzierba A R 2001 *hep-ex/0106010*

[3] ISO 2012 *ISO/IEC 14882:2011 Information technology — Programming languages — C++* (Geneva, Switzerland: International Organization for Standardization) URL `http://www.iso.org/iso/iso_catalogue/catalogue_tc/catalogue_detail.htm?csnumber=50372`

[4] Brigljevic V, Bruno G, Cano E, Cittolin S, Csilling A, Gigi D, Glege F, Gomez-Reino R, Gulmini M, Gutleber J, Jacobs C, Kozlovszky M, Larsen H, de Abril I M, Meijers F, Meschi E, Murray S, Oh A, Orsini L, Pollet L, Racz A, Samyn D, Scharff-Hansen P, Schwick C, Sphicas P, ODell V, Suzuki I, Berti L, Maron G, Toniolo N, Zangrando L, Ninane A, Erhan S, Bhattacharya S and Branson J 2003 The cms event builder (*Preprint* `arXiv:physics/0306150`)

[5] Andre J M, Andronidis A, Behrens U, Branson J, Brummer P, Chaze O, Contescu C, Craigs B G, Cittolin S, Darlea G L, Deldicque C, Demiragli Z, Dobson M, Erhan S, Fulcher J R, Gigi D, Glege F, Gomez-Ceballos G, Hegeman J, Holzner A, Jimenez-Estupianan R, Masetti L, Meijers F, Meschi E, Mommsen R K, Morovic S, ODell V, Orsini L, Paus C, Pieri M, Racz A, Sakulin H, Schwick C, Reis T, Simelevicius D and Zejdl P 2016 Performance of the new DAQ system of the CMS experiment for run-2 *2016 IEEE-NPSS Real Time Conference (RT)* (IEEE) URL `https://doi.org/10.1109/rtc.2016.7543164`